Answers

Storage parallelism and skew:

a. III

Explanation: It's possible that multiple objects will have the same price, causing attribute-value skew. It is possible that certain objects will be accessed more frequently than others, causing execution skew.

b. VII

Explanation: On top of attribute value and execution skew (see previous question), we also need to worry about the fact that there is a very small number of objects, so it's actually very likely that for example at least one server will have 2 data objects, causing partition skew.

c. VII

Explanation: On top of attribute value and execution skew, the products might not be evenly distributed in price (e.g., all of them might be priced between $50-$70) causing a partition skew.

d. V

Explanation: With a million objects across 10 nodes, the probability that a hash partition will cause a data skew is close to zero. So the only causes of skew will be attribute skew + execution skew, similar to question a.

e. III

Explanation: since each name is unique, there is no fear of attribute-value skew (since there cannot be duplicate attributes). We should still be worried about execution skew, since we do not know how the objects are accessed (e.g., a certain object might be accessed much more than others).

f. III

Explanation: same as e. Since we are using 1 million objects across 10 nodes, we do not need to worry about data skew for hashing.

g. VI

Explanation: Partition skew can happen, since all product names might start with the letter "a". Attribute-value skew is impossible since all names are unique, and execution skew is possible since we do not know the workload request distribution.

h. VIII

Explanation: we do not need to worry about skew, since product names are unique (no attribute-value skew), hashing 1 million objects over 10 servers will not produce partitioning skew, and since we know the access distribution is totally uniform across the objects, we will not have execution skew.

Replication

1. II.
Explanation: The best possible write latency in this scheme is when we write to the primary replica (10 ms), and in parallel it writes to the two other nodes (10 ms), for a total of 20ms
2. I.
Explanation: a read always goes to the master replica, and the latency is 10ms.
3. III.
Explanation: The latency is the time it takes for the first split to be written to the primary node, plus the amount of time it takes to write the entire data block to the two secondary nodes (in parallel).
4.
Advantage: the write latency is lower (10ms). There are several disadvantages. First, if the master replica dies before it replicates to the secondaries, the latest write will be lost. Second, if the replication hasn't completed writing to the secondary nodes, the application may read a stale copy of the data. Third, if the master replica dies, it may have replicated its data to only one node (and not the other), and it's not clear which node has the latest copy.

HDFS

1. Using very small files means that there would be a proportionally a much larger number of files per DataNode. The downside in that case is that the NameNode would have to keep mappings of a much larger number of files, which would result in a very large index, which would exceed the amount of memory on the NameNode and make it become a bottleneck.
2. The application could see stale/old data. For example, there could be a network partition separating the DataNode from the NameMode, which leads the NameNode to believe the DataNode is unavailable and remap its data to another node, which continues accepting writes. But the application might try to access the old DataNode directly and not see the latest writes to files.

2PC

1. P3 has no <ready T1> in log, so the system should abort T1.

2. After sending <abort T1> to P1.

3. Consult the coordinator for an answer about T1's state.

4. P1 and P2 would log <commit T1> and commit T1. P3 would keep waiting until it reconnects with the coordinator.

MapReduce

1) We can accept two answers:
   Map(Tom Jerry Jerry) = {‘Tom’: 1, ‘Jerry’: 1, ‘Jerry’: 1}
   Map(Tom Spike Tom) = {‘Tom’: 1, ‘Spike’: 1, ‘Tom’: 1}
   Map(Jerry Tom Jerry) = {‘Jerry’: 1, ‘Tom’: 1, ‘Jerry’: 1}
   Map(Tom Tom Jerry) = {‘Tom’: 1, ‘Tom’: 1, ‘Jerry’: 1}
   Map(Jerry Jerry Jerry) = {‘Jerry’: 1, ‘Jerry’: 1, ‘Jerry’: 1}

   or:

   Map(Tom Jerry Jerry) = {‘Tom’: 1, ‘Jerry’: 2}
   Map(Tom Spike Tom) = {‘Tom’: 2, ‘Spike’: 1}
   Map(Jerry Tom Jerry) = {‘Jerry’: 2, ‘Tom’: 1}
   Map(Tom Tom Jerry) = {‘Tom’: 2, ‘Jerry’: 1}
   Map(Jerry Jerry Jerry) = {‘Jerry’: 3}

2)
   Reduce({‘Tom’: [1, 2, 1, 2]}) = {‘Tom’: 6}
   Reduce({‘Jerry’: [2, 2, 1, 3]}) = {‘Jerry’: 8}
   Reduce({‘Spike’: [1]}) = {‘Spike’: 1}

Caching

1: FIFO
1 0 4 3, total number of misses = 4
2 1 0 4, total number of misses = 5
3 2 1 0, total number of misses = 6
4 3 2 1, total number of misses = 7
5 4 3 2, total number of misses = 8
Hit rate: 9 / 17

2: LRU
1 0 4 3, total number of misses = 4
2 1 0 4, total number of misses = 5
3 2 1 4, total number of misses = 6
5 1 2 4, total number of misses = 7
Hit rate: 10 / 17

3. LRU

4. iii. Both miss each and every request.

5. vii, 75%. After the first four accesses, Belady's cache will be: 4, 3, 2, 1. The following requests will lead to:

5 → 5, 3, 2, 1 (miss)
1 → 1, 5, 2, 3 (hit)
2 → 2, 1, 5, 3 (hit)
3 → 3, 2, 1, 5 (hit)
4 → 4, 2, 1, 5 (miss)
5 → 5, 4, 2, 1 (hit)
1 → 1, 5, 4, 2 (hit)
2 → 2, 1, 5, 4 (hit)
….
So for every 3 hits, 1 miss, and a total of 75%.

6. v. 40%. The odd cache will miss all its requests (since it sees the pattern 1, 3, 5, 1, 3, 5, … with only 2 slots), while the even cache will hit all its requests (since it sees the pattern 2, 4, 2, 4 with 2 slots). So the total hit rate will be 40%.

Tensorflow

1: B

2: B

3: A

Security goals

1. iii. The attacker is trying to take down the server, which would cause unavailability.
2. ii. The attacker is breaking the integrity of the system, by moving money from account to the other without authorization.
3. iii. The attackers are trying to make the government website unavailable.
4. iv. The attackers are illegitimately impersonating a trusted source.
5. i. The passwords are confidential information.
6. ii. The attacker is tampering with the normal function of the database.
7. i. The pictures are confidential, and the attacker is breaking that confidentiality.
8. iv. The foreign government is faking a trusted source (scientific studies).

Encryption

1. ii + iv: a signature provides integrity (to ensure the email wasn't tampered with) and authenticity (to verify the identity of the person who sent it)
2. i + ii + iv: the dropbox.com signature provides integrity and authenticity. The encryption of the file with the user's symmetric key ensures confidentiality (that no one else will be able to open it).
3. ii: the hash function ensures integrity: if the data is tampered with, the output of the hash function would not match the locally saved output of when the file was written.
4. The new design decreases the security of the system, because it no longer provides integrity. The attacker can tamper with the data, and then generate a new hash and replace the existing hash on the data store with the new hash, and it will appear as if the data wasn't tampered with.

Compliance

1. Auditing is important because it allows you to keep track of who accessed the data (and see that no one unauthorized accessed it), and if there is a breach it could help us investigate how it happened.
2. No. It does not ensure availability. One way to improve that is to add some additional replicas or put it on a cloud provider that gives DDoS protection.
It also does not address integrity, since there is no way to know if someone tampered with the data.
One option to address this would be to hash each entry and keep the hashes in a secure store, which can be used to verify if the data was tampered with. Another simpler solution, is to keep a log not just of which user logged into the system, but also log each change to the database and who made the change, which would allow you to monitor and rollback any changes to the sensitive data.